UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/894,096	06/28/2001	Michael Albert Perks	AUS920010171US1	5339

7590 02/18/2005

Joseph R. Burwell
Law Office of Joseph R. Burwell
P.O. Box 28022
Austin, TX 78755-8022

EXAMINER	
MOFIZ, APU M	

ART UNIT	PAPER NUMBER
2165	

DATE MAILED: 02/18/2005

Please find below and/or attached an Office communication concerning this application or proceeding.

Office Action Summary	Application No.	Applicant(s)	
	09/894,096	PERKS ET AL.	
	Examiner	Art Unit	
	Apu M Mofiz	2165	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *16 August 2004*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-30* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☒ Claim(s) *12,13,25,26 and 30* is/are allowed.

6)☒ Claim(s) *1-8,14-21 and 27-29* is/are rejected.

7)☒ Claim(s) *9-11 and 22-24* is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *28 June 2001* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application (PTO-152)
6) ☐ Other: _____.

DETAILED ACTION

Examiner's Response to Applicant's Remarks

1. Applicant's arguments submitted on 08/16/2004 with respect to claims 1-30 have been reconsidered but are not deemed persuasive for the reasons set forth below.

Examiner's Responses to Applicant's Remarks are listed below:

2. Applicant argues (under REMARKS section) that, Heistermann et al. does not teach "associating an attribute version identifier with an attribute in the set of attributes such that each attribute in the set of attributes is associated with an attribute version identifier."

Examiner respectfully disagrees. Heistermann teaches **associating an attribute version identifier** (i.e. ***"Step 121 identifies the properties for this program object version that should be included in the serialized representation of the program object." ... "In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."*** ... *"If this object-descriptor base class, the process continues with step 133 that writes to the serial information stream an identification of the version pertaining to the base class. The process continues by serializing the appropriate properties of that base class.";* The preceding text excerpts clearly indicate that when an object is serialized, the corresponding class version identifier is also serialized. With the serialization of the object, the properties or attributes also have to be identified to be serialized so that the object can be de-serialized and recreated later. But the corresponding class for the object can be inherited from a base class. In that case the object is going to inherit attributes from the base class. The inherited attributes or properties will have to be identified differently. Some of the attributes can be serialize-able objects themselves. Therefore, **Heistermann says that the identifications of the attributes/properties are a function of the program object class and version**. Let us say that the class version of the program object is X2, which is inherited from the base class whose version is X1. The

properties/attributes of program object class are identified as y2=f(x2) and the properties inherited from the base class to the inherited program object class is y1=f(x1). Therefore each attribute in the set of attributes for the program object class is either associated with f(x1) or f(x2) identifiers. The applicant never claims that the attribute/property identifier may not be associated with the class version identifier.) (col 6, lines 38-53; col 7, lines 35-46) **with an attribute** (i.e. a property) (col 6, lines 38-53; col 7, lines 35-46) **in the set of attributes** (i.e. a set of properties of a class) (col 6, lines 38-53; col 7, lines 35-46) **such that each attribute in the set of attributes is associated with an attribute version identifier** (i.e., **Heistermann says that the identifications of the attributes/properties are a function of the program object class and version.** Let us say that the class version of the program object is X2, which is inherited from the base class whose version is X1. The properties/attributes of program object class are identified as y2=f(x2) and the properties inherited from the base class to the inherited program object class is y1=f(x1). Therefore each attribute in the set of attributes for the program object class is either associated with f(x1) or f(x2) identifiers. The applicant never claims that the attribute/property identifier may not be associated with the class version identifier.) (col 6, lines 38-53; col 7, lines 35-46).

Any other arguments by the applicant are more limiting than the claimed language.

3. Applicant is inaccurate for the reasons explicitly stated in the First Office Action. Examiner asserts that the Heistermann teaches Applicant's invention.

4. These reasons have been explicitly stated in the First Office Action. Please see the next section.

Claim Rejections - 35 USC § 102

5. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

> A person shall be entitled to a patent unless –
>
> (e) the invention was described in (1) an application for patent, published under section 122(b), by another filed in the United States before the invention by the applicant for patent or (2) a patent granted on an application for patent by another filed in the United States before the invention by the applicant for patent, except that an international application filed under the treaty defined in section 351(a) shall have the effects for purposes of this subsection of an application filed in the United States only if the international application designated the United States and was published under Article 21(2) of such treaty in the English language.

6. Claims 1-8, 14-21, and 27-29 are rejected under 35 U.S.C. 102(e) as being anticipated by Heistermann et al. (U.S. Patent No. 6,477,701 and Heistermann hereinafter).

As to claim 1, Heistermann teaches a method for object-oriented management of serializable objects (i.e. serializing and de-serializing program objects) (col 3, lines 1-22), the method comprising: identifying an object (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40), wherein the object comprises a set of attributes (i.e. properties) (col 6, lines 32-53); associating a class version identifier with the object (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62), wherein the class version identifier identifies the object as an instance of a specific version of a class (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40); and associating an attribute version identifier (i.e. ***"Step 121 identifies the properties for this program object version that should be included in the serialized representation of the program object." ... "In any case, an implementation of a process that determines***

the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122." ... *"If this object-descriptor base class, the process continues with step 133 that writes to the serial information stream an identification of the version pertaining to the base class. The process continues by serializing the appropriate properties of that base class.";* The preceding text excerpts clearly indicate that when an object is serialized, the corresponding class version identifier is also serialized. With the serialization of the object, the properties or attributes also have to be identified to be serialized so that the object can be de-serialized and recreated later. But the corresponding class for the object can be inherited from a base class. In that case the object is going to inherit attributes from the base class. The inherited attributes or properties will have to be identified differently. Some of the attributes can be serialize-able objects themselves. Therefore, **Heistermann says that the identifications of the attributes/properties are a function of the program object class and version**. Let us say that the class version of the program object is X2, which is inherited from the base class whose version is X1. The properties/attributes of program object class are identified as y2=f(x2) and the properties inherited from the base class to the inherited program object class is y1=f(x1). Therefore each attribute in the set of attributes for the program object class is either associated with f(x1) or f(x2) identifiers. The applicant never claims that the attribute/property identifier may not be associated with the class version identifier.) (col 6, lines 38-53; col 7, lines 35-46) with an attribute (i.e. a property) (col 6, lines 38-53; col 7, lines 35-46) in the set of attributes (i.e. a set of properties of a class) (col 6, lines 38-53; col 7, lines 35-46) such that each attribute in the set of attributes is associated with an attribute version identifier.

As to claim 2, Heistermann teaches that an attribute version identifier (i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53) represents an instance of a specific version of a class (i.e. *"In any case, an implementation of a process that*

determines the properties to serialize may be considered a function of the program object class and version.) (col 6, lines 38-53) in which the associated attribute was initially declared within the class (i.e. *"In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."*; i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53).

As to claim 3, Heistermann teaches writing a data stream (i.e. serial stream of information) (col 6, lines 38-53) representing an object serialization (i.e. *"According to this process, step 131 writes the number of program objects to serialize into a serial stream of information."*) (col 6, lines 54-62) of the object (col 6, lines 38-53), wherein the data stream (col 6, lines 38-53) comprises the class version identifier of the object (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40), an attribute value (i.e. the properties value) (col 6, lines 38-53) for an attribute in the set of attributes (i.e. the properties of the object) (col 6, lines 38-53), and an attribute version identifier for an attribute in the set of attributes (i.e. *"In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."*) (col 6, lines 38-53).

As to claim 4, Heistermann teaches writing a class identifier (i.e. *"String className = in.readUTF();"*) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67); and instantiating the object in accordance with the class identifier (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

String className = in.readUTF(); //Read object class name;

...

obj = objClass.newInstance();

...") (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) for the class of the object into the data stream (i.e. *"According to the illustrated format, segment 301 provides an indication of the number of program objects represented by information in the stream, segments 302 and 303 provide an indication of the class name and version of the first program object represented in the stream, respectively, and segment 304 conveys serial information representing one or more properties of the first program object."*) (col 8, lines 15-24).

As to claim 5, Heistermann teaches writing an attribute count (i.e. *props.length*) (col 11, lines 1-25; col 14, lines 37-67) indicating a number of attributes (i.e. properties) (col 11, lines 1-25; col 14, lines 37-67) from the set of attributes that were written into the data stream (i.e.

"private void serializeObject(ObjectOutputStream out, Xserializable obj){

...

...

PropertyDescriptor[] props = descriptor.getProperties(version);

For (int j=0; j< props.length; j++) { ..."

...

"private Object deserializeObject(ObjectInputStream in) {

...

PropertyDescriptor[] props = objDesc.getProperties(version);

...

for (int j=0; j< props.length; j++) { ... "; i.e. the above code excerpts shows that the properties count/length is also stored in the stream. The de-serialization method retrieves the properties count from the stream passed in the method parameter.) (col 11, lines 1-25; col 14, lines 37-67).

As to claim 6, Heistermann teaches reading a data stream (i.e. serial stream of information) (col 6, lines 38-53) representing a serialized object (i.e.

"private void serializeObject(ObjectOutputStream out, Xserializable obj){...

...

out.writeObject(retVal); //Write object"; i.e. the above code excerpts shows the serialization of a object) (col 11, lines 1-25; col 14, lines 37-67), wherein the data stream (col 6, lines 38-53) comprises a serialized class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62), a set of serialized attribute values (i.e. properties values) (col 11, lines 1-25; col 14, lines 37-67), and a set of serialized attribute version identifiers (i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53), wherein serialized attribute version identifiers (col 6, lines 38-53) in the set of serialized attribute version identifiers (col 6, lines 38-53) are paired with serialized attribute values (i.e. properties values) (col 11, lines 1-25; col 14, lines 37-67) in the set of serialized attribute values (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

float version = in.readFloat();

...

PropertDescriptor[] props = objDesc.getProperties(version);

...

for (int j=0; j< props.length; j++) {

...

propVal = deserializeObject(in);

...

return obj;

}"; i.e. the above code excerpts shows that the de-serialization method gets the data stream as the input parameter. From the stream the method retrieves the class version and associated properties/attributes version. Then the method goes on retrieving the property value. Therefore the properties version and values are stored in the same place i.e. block 304 in Fig. 4) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67).

As to claim 7, Heistermann teaches reading a class identifier (i.e. the class name) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) for the serialized object (i.e. *ObjectInputStream*) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) from the data stream (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

String className = in.readUTF(); //Read object class name;

... ") (Fig. 4; col 11, lines 1-25; col 14, lines 37-67); and instantiating the object in accordance with the class identifier (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

String className = in.readUTF(); //Read object class name;

...

obj = objClass.newInstance(); ... ") (Fig. 4; col 11, lines 1-25; col 14, lines 37-67), wherein the class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62) of the object and the serialized class version identifier (col 6, lines 54-62) of the serialized object (i.e. *ObjectInputStream*) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) may differ (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

String className = in.readUTF(); //Read object class name;

...

Class objClass = Class.forName(desc.getClassName());

obj = objClass.newInstance(); ... " ; i.e. the above code excerpts shows that the new object is instantiated by just using the class name, which is retrieved from the input data stream; At this point the new instance of the object is not aware of the class version of the serialized object and therefore may very well differ from the serialized class version.) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67).

As to claim 8, Heistermann teaches reading an attribute count (i.e. *props.length*) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) for the set of serialized attribute values (properties values) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) from the data stream (i.e. serial stream of information) (col 6, lines 38-53).

As to claim 14, Heistermann teaches a computer program product on a computer readable medium for use in a data processing system for object-oriented management

of serializable objects (i.e. serializing and de-serializing program objects) (col 3, lines 1-22), the computer program product comprising: instructions for identifying an object (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40), wherein the object comprises a set of attributes (i.e. properties) (col 6, lines 32-53); instructions for associating a class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62) with the object (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62), wherein the class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62) identifies the object as an instance of a specific version of a class (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40); and instructions for associating an attribute version identifier (i.e. ***"Step 121 identifies the properties for this program object version that should be included in the serialized representation of the program object." ... "In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."*** ... *"If this object-descriptor base class, the process continues with step 133 that writes to the serial information stream an identification of the version pertaining to the base class. The process continues by serializing the appropriate properties of that base class."*; The preceding text excerpts clearly indicate that when an object is serialized, the corresponding class version identifier is also serialized. With the serialization of the object, the properties or

attributes also have to be identified to be serialized so that the object can be de-serialized and recreated later. But the corresponding class for the object can be inherited from a base class. In that case the object is going to inherit attributes from the base class. The inherited attributes or properties will have to be identified differently. Some of the attributes can be serialize-able objects themselves. Therefore, **Heistermann says that the identifications of the attributes/properties are a function of the program object class and version.** Let us say that the class version of the program object is X2, which is inherited from the base class whose version is X1. The properties/attributes of program object class are identified as y2=f(x2) and the properties inherited from the base class to the inherited program object class is y1=f(x1). Therefore each attribute in the set of attributes for the program object class is either associated with f(x1) or f(x2) identifiers. The applicant never claims that the attribute/property identifier may not be associated with the class version identifier.) (col 6, lines 38-53; col 7, lines 35-46) with an attribute (i.e. a property) (col 6, lines 38-53; col 7, lines 35-46) in the set of attributes (i.e. a set of properties of a class) (col 6, lines 38-53; col 7, lines 35-46) such that each attribute in the set of attributes is associated with an attribute version identifier.

As to claim 15, Heistermann teaches that an attribute version identifier (i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53) represents an instance of a specific version of a class (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62) in which the associated attribute (col 6, lines 38-53; col 7, lines 35-46) was initially declared (i.e. the attribute/property was declared in the class) (col 6, lines 38-53; col 7, lines 35-46) within the class (i.e. *"In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective*

property that was identified in steps 121 and 122."; i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53).

As to claim 16, Heistermann teaches instructions for writing a data stream (i.e. serial stream of information) (col 6, lines 38-53) representing an object serialization of the object (col 6, lines 38-53), wherein the data stream (col 6, lines 38-53) comprises the class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62) of the object (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40), an attribute value (i.e. the properties value) (col 6, lines 38-53) for an attribute in the set of attributes (i.e. the properties of the object) (col 6, lines 38-53), and an attribute version identifier (i.e. *"Step 121 identifies the properties for this program object version that should be included in the serialized representation of the program object."* ... *"In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."* ... *"If this object-descriptor base class, the process continues with step 133 that writes to the serial information stream an identification of the version pertaining to the base class. The process continues by serializing the appropriate properties of that base class."*; i.e. whether the properties/attributes are from a child class or a base class they are the function of the program object class and version. Therefore properties from a child class object would have different identifier than the base class properties and both sets of properties would be serialized with different identifiers) (col 6, lines 38-53; col 7, lines 35-46) for an attribute in the set of attributes (i.e. *"In any case, an implementation of a process that determines the properties to serialize may be considered a*

function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122.") (col 6, lines 38-53).

As to claim 17, Heistermann teaches instructions for writing a class identifier (i.e. the class name) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) for the class of the object into the data stream (i.e. *"According to the illustrated format, segment 301 provides an indication of the number of program objects represented by information in the stream, segments 302 and 303 provide an indication of the class name and version of the first program object represented in the stream, respectively, and segment 304 conveys serial information representing one or more properties of the first program object."*) (col 8, lines 15-24).

As to claim 18, Heistermann teaches instructions for writing an attribute count (i.e. *props.length*) (col 11, lines 1-25; col 14, lines 37-67) indicating a number of attributes from the set of attributes that were written into the data stream (i.e.

"private void serializeObject(ObjectOutputStream out, Xserializable obj){

...

...

PropertDescriptor[] props = descriptor.getProperties(version);

For (int j=0; j< props.length; j++) { ... "

...

"private Object deserializeObject(ObjectInputStream in) {

...

PropertDescriptor[] props = objDesc.getProperties(version);

...

for (int j=0; j< props.length; j++) { ... "; i.e. the above code excerpts shows that the properties count/length is also stored in the stream. The de-serialization method retrieves the properties count from the stream passed in the method parameter.) (col 11, lines 1-25; col 14, lines 37-67).

As to claim 19, Heistermann teaches instructions for reading a data stream (i.e. serial stream of information) (col 6, lines 38-53) representing a serialized object (i.e.

"private void serializeObject(ObjectOutputStream out, Xserializable obj){...

...

out.writeObject(retVal); //Write object"; i.e. the above code excerpts shows the serialization of a object) (col 11, lines 1-25; col 14, lines 37-67), wherein the data stream (col 6, lines 38-53) comprises a serialized class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62), a set of serialized attribute values (i.e. properties values) (col 11, lines 1-25; col 14, lines 37-67), and a set of serialized attribute version identifiers (i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53), wherein serialized attribute version identifiers (col 6, lines 38-53) in the set of serialized attribute version identifiers (col 6, lines 38-53) are paired with serialized attribute values (i.e. properties values) (col 11, lines 1-25; col 14, lines 37-67) in the set of serialized attribute values (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

float version = in.readFloat();

...

PropertDescriptor[] props = objDesc.getProperties(version);

...

for (int j=0; j< props.length; j++) {

...

propVal = deserializeObject(in);

...

return obj;

}"; i.e. the above code excerpts shows that the de-serialization method gets the data stream as the input parameter. From the stream the method retrieves the class version and associated properties/attributes version. Then the method goes on retrieving the property value. Therefore the properties version and values are stored in the same place i.e. block 304 in Fig. 4) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67).

As to claim 20, Heistermann teaches instructions for reading a class identifier (i.e. the class name) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) for the serialized object from the data stream (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

String className = in.readUTF(); //Read object class name; ... ") (Fig. 4; col 11, lines 1-25; col 14, lines 37-67); and instructions for instantiating the object in accordance with the class identifier (i.e. *"private Object deserializeObject(ObjectInputStream in) {*

...

String className = in.readUTF(); //Read object class name;

...

obj = objClass.newInstance(); ... ") (Fig. 4; col 11, lines 1-25; col 14, lines 37-67), wherein the class version identifier of the object and the serialized class version identifier of the serialized object may differ (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

String className = in.readUTF(); //Read object class name;

...

Class objClass = Class.forName(desc.getClassName());

obj = objClass.newInstance(); ... " ; i.e. the above code excerpts shows that the new object is instantiated by just using the class name, which is retrieved from the input data stream; At this point the new instance of the object is not aware of the class version of the serialized object and therefore may very well differ from the serialized class version.) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67).

As to claim 21, Heistermann teaches instructions for reading an attribute count (i.e. *props.length*) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67) for the set of serialized attribute values from the data stream (Fig. 4; col 11, lines 1-25; col 14, lines 37-67).

As to claim 27, Heistermann teaches an apparatus for object-oriented management of serializable objects (i.e. serializing and de-serializing program objects) (col 3, lines 1-22), the apparatus comprising: means for identifying an object (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40), wherein the object comprises a set of attributes (i.e. properties) (col 6, lines 32-53); means for associating a class version identifier with the object (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an*

identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object.") (col 6, lines 54-62), wherein the class version identifier identifies the object as an instance of a specific version of a class (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40); and means for associating an attribute version identifier (i.e. ***"Step 121 identifies the properties for this program object version that should be included in the serialized representation of the program object." ... "In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."*** ... *"If this object-descriptor base class, the process continues with step 133 that writes to the serial information stream an identification of the version pertaining to the base class. The process continues by serializing the appropriate properties of that base class.";* The preceding text excerpts clearly indicate that when an object is serialized, the corresponding class version identifier is also serialized. With the serialization of the object, the properties or attributes also have to be identified to be serialized so that the object can be de-serialized and recreated later. But the corresponding class for the object can be a inherited from a base class. In that case the object is going to inherit attributes from the base class. The inherited attributes or properties will have to be identified differently. Some of the attributes can be serialize-able objects themselves. Therefore, **Heistermann says that the identifications of the attributes/properties are a function of the program object class and version**. Let us say that the class version of the program object is X2, which is inherited from the base class whose version is X1. The properties/attributes of program object class are identified as y2=f(x2) and the properties inherited from the base class to the inherited program object class is y1=f(x1). Therefore each attribute in the set of attributes for the program object class is either associated with f(x1) or f(x2) identifiers. The applicant never claims that the attribute/property identifier may not be associated with the class version identifier.) (col 6, lines 38-53; col 7, lines 35-46) with an attribute (i.e. a property) (col 6, lines 38-53; col 7, lines 35-46) in the set of attributes (i.e. a set of properties of a class)

(col 6, lines 38-53; col 7, lines 35-46) such that each attribute in the set of attributes is associated with an attribute version identifier.

As to claim 28, Heistermann teaches means for writing a data stream (i.e. serial stream of information) (col 6, lines 38-53) representing an object serialization of the object (col 6, lines 38-53), wherein the data stream comprises the class version identifier of the object (i.e. *"Step 101 instantiates a program object of this class and step 110 invokes the appropriate method to obtain an identification of the program object version."*) (col 6, lines 35-40), an attribute value (i.e. the properties value) (col 6, lines 38-53) for an attribute in the set of attributes, and an attribute version identifier for an attribute in the set of attributes (i.e. *"In any case, an implementation of a process that determines the properties to serialize may be considered a function of the program object class and version. Step 122 ascertains the value of each property to serialize and step 130 generates serial information that conveys the program object class and version, and a representation of each respective property that was identified in steps 121 and 122."*) (col 6, lines 38-53).

As to claim 29, Heistermann teaches means for reading a data stream (i.e. serial stream of information) (col 6, lines 38-53) representing a serialized object (i.e.

"private void serializeObject(ObjectOutputStream out, Xserializable obj){...

...

out.writeObject(retVal); //Write object" ; i.e. the above code excerpts shows the serialization of a object) (col 11, lines 1-25; col 14, lines 37-67), wherein the data stream (col 6, lines 38-53) comprises a serialized class version identifier (i.e. *"For each respective program object to serialize, step 132 writes into the serial information stream an identification of the class of the respective program object and step 133 writes an identification of the version of the respective program object."*) (col 6, lines 54-62), a set of serialized

attribute values (i.e. properties values) (col 11, lines 1-25; col 14, lines 37-67), and a set of serialized attribute version identifiers (i.e. the properties version identifier is a function of the program object class and version. The property identifier corresponds to the version of the class it belongs to or was declared in.) (col 6, lines 38-53), wherein serialized attribute version identifiers (col 6, lines 38-53) in the set of serialized attribute version identifiers (col 6, lines 38-53) are paired with serialized attribute values (i.e. properties values) (col 11, lines 1-25; col 14, lines 37-67) in the set of serialized attribute values (i.e.

"private Object deserializeObject(ObjectInputStream in) {

...

float version = in.readFloat();

...

PropertDescriptor[] props = objDesc.getProperties(version);

...

for (int j=0; j< props.length; j++) {

...

propVal = deserializeObject(in);

...

return obj;

}" ; i.e. the above code excerpts shows that the de-serialization method gets the data stream as the input parameter. From the stream the method retrieves the class version and associated properties/attributes version. Then the method goes on retrieving the property value. Therefore the properties version and values are stored in the same place i.e. block 304 in Fig. 4) (Fig. 4; col 11, lines 1-25; col 14, lines 37-67).

Allowable Subject Matter

7. Claims 12-13,25-26 and 30 are allowed over the prior art of record.

As to claims 12-13,25-26 and 30, Heistermann does not disclose, teach or suggest the claimed limitations of (in combination with all other features in the claims), a method and associated system for providing backwards and forwards compatibility between different versions of serialized object data, which includes identifying an object, wherein the object comprises a set of attributes, wherein each attribute in the set of attributes is associated with a version identifier, and wherein the object is an instance of a first version of a class; writing a data stream representing serialization of the object's attributes and associated version identifiers; reading a data stream representing a serialized object into a new object instance of a second version of a class; and refraining from storing attributes from the data stream into the new object instance that are not represented in the new object instance while reading the data stream.

8. Claims 9-11 and 22-24 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten in independent form including all of the limitations of the base claim and any intervening claims.

The following is a statement of reasons for the indication of allowable subject matter:

As to claims 9-11 and 22-24, the prior art of record Heistermann et al. (U.S. Patent No. 6,477,701 and Heistermann hereinafter) does not disclose, teach or suggest the claimed limitations of (in combination with all other features in the claims), mapping

attributes between the object and the serialized object; and storing serialized attribute values from the data stream in the object.

The closest prior arts fail to anticipate or render Applicant's limitations above obvious.

Conclusion

9. **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

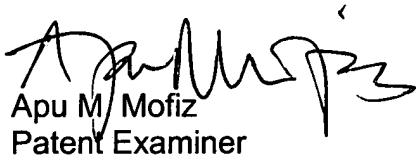
A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Points of Contact

10. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Apu M. Mofiz whose telephone number is (571) 272-4080. The examiner can normally be reached on Monday – Thursday 8:00 A.M. to 4:30 P.M.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Dov Popovici can be reached at (571) 272-4083. The fax numbers for the group is (703) 872-9306.

Any inquiry of a general nature or relating to the status of this application should be directed to the Group receptionist whose telephone number is (703) 305-9600.



Apu M. Mofiz
Patent Examiner
Technology Center 2100

February 15,2004